Слайд 2

Для хранения информации программы обычно используют переменные – именованные области компьютерной памяти, в которых можно хранить различные значения. В языке Си, прежде чем использовать переменную, ее надо *объявить*. Сделать это нужно *до того, как переменная будет использована*. Оператор объявления выглядит следующим образом:

```
int age;
```

где `int` – тип переменной, говорящий компилятору, какую информацию можно будет в ней хранить. В нашем случае это `int` (integer) – целочисленный тип. После типа идет имя переменной (в нашем примере – `age`), которое может быть *любой комбинацией букв, цифр и символа нижнего подчеркивания, начинающейся с буквы или символа нижнего подчеркивания.*

Слайд 3

Для ввода информации с клавиатуры используется еще одна функция из библиотеки `stdio` – `scanf_s`:

```
scanf_s("%d", &age);
```

Первый параметр этой функции – форматная строка, которая описывает, что и какого типа нужно считать с клавиатуры. `%d` означает, что надо считать одно десятичное число (decimal). Второй параметр – имя переменной, куда функция должна сохранить введенное с клавиатуры значение, со специальным символом `&`, предназначение которого мы поймем чуть позже.

Слайд 4

```
1. #include <stdio.h>
2. #include <locale.h>
3. int main(){
4. setlocale(LC_ALL, "Russian");
5. int age, my_age;
6. printf("Сколько вам лет?\n");
7. scanf_s("%d", &age);
8. my_age = age + 2;
9. printf("Ваш возраст: %d. Но я старше! Мой возраст –
%d!!!\n", age, my_age);
10. return 0;
11. }
```

Функция scanf_s не является обязательной частью стандарта языка Си и на настоящий момент полностью поддерживается лишь в Microsoft Visual Studio. По мнению Microsoft, scanf_s является безопасной версией функции scanf (отсюда суффикс _s – secure).

Мы в одном операторе объявили одновременно сразу несколько переменных (строка 5). В-третьих – мы познакомились с оператором *присваивания* (строка 8), который кладет (присваивает) значение правой части в переменную, имя которой указано в левой части. В этой же строке мы видим пример простейшего выражения, вычисляющего сумму. Ну и, наконец, в-пятых, вывод на печать значений переменных в строке 10.

Слайд 5

Переменные и константы – два механизма для работы с данными в любом языке программирования. *Переменная* – это именованная область памяти, используемая для хранения промежуточных результатов. Ее содержимое может изменяться, откуда и название – переменная. *Константа* же (лат. *constanta* – постоянная, неизменная) – это некоторое конкретное значение (числовое или строковое), которое, будучи определенным в программе, меняться уже не может.
Константы бывают двух типов: *литеральные* и *символические*.

Слайд 6

Литеральные константы (или просто литералы) – это *конкретные значения*, записанные непосредственно в тексте программы. Они бывают:
– числовыми (целочисленными и дробными), например: `0, -5, 3.14, 103` и т.д.,
– строковыми (заключенными в двойные кавычки) – `"Hello, world!\n"`, `"Input a number: "` и т.д.,
– символьными (заключенными в одинарные кавычки) – `'a', 'b', '0'`, и т.д.

Слайд 7

Целочисленные константы в языке Си могут быть представлены в одной из следующих систем счисления (перед любой формой записи может идти знак плюс или минус):
– десятичные: последовательность цифр от `0` до `9`, которая начинается с цифры, отличной от `0` (исключение составляет лишь сам `0`). Пример: `2, -45, 23`;
– восьмеричные: последовательность цифр от `0` до `7`, которая всегда начинается с нуля. Пример: `00, 032, 063`;
– шестнадцатеричные: последовательность цифр от `0` до `9` и символов от `A` до `F` (без учета регистра), которая всегда начинается с комбинации `0x`. Пример: `0x0, 0x4, 0xFFFF, 0x2A4, 0x7fff`.

Слайд 8

Символические константы – это своего рода «переменные», значения которых меняться не могут. Их также необходимо объявлять в коде программы, но перед типом надо ставить ключевое слово `const`. Например:

```
const double pi = 3.14;
```

При этом очень важно помнить, что присваивать значение символической константе необходимо *одновременно с ее объявлением*. Код вида:

```
const double pi;
pi = 3.14;
```

будет некорректным, потому что ключевое слово `const` говорит компилятору языка Си, что значение «переменной» `pi` после ее создания измениться уже не может.

Слайд 9

Переменные бывают разных типов. Си – язык со *статической типизацией*. Это означает, что любая переменная должна иметь конкретный тип, и она сможет хранить в себе данные только этого типа (в отличие от, например, Бейсика, Питона и ряда других языков, где одно и то же имя можно использовать для поочередного хранения значений самых разных типов). В языке Си есть следующие базовые (включенные в синтаксис самого язык) типы данных:

– char – символ,
– int – целое число,
– float – дробное число одинарной точности,
– double – дробное число двойной точности.

Создаются переменные при помощи *оператора объявления*, который выглядит в общем виде так:

тип имя$_1$ [= значение$_1$], …, имя$_N$ [= значение$_N$];

Этот код позволяет нам *инициализировать* (присвоить начальное значение) переменным прямо во время создания. Если этого не сделать, то после создания переменная будет содержать «мусор» – значение, которое заранее неизвестно программисту и зависит от настроения компилятора и операционной системы.

Для записи в переменную какого-либо значения, используется *оператор присваивания* (=). Этот оператор работает следующим образом:

куда = что;

Функция printf на строке 10 печатает фразу, в которую вставлены значения соответствующих переменных. В данном случае используются те же спецификаторы %d, что и в scanf_s. Напомним, что спецификатор состоит из двух символов, первый – %, второй – буква латинского алфавита, показывающая, со значением какого типа предстоит работать (d – decimal, десятичное). На места, занимаемые спецификаторами в форматной строке,

вставляются по порядку значения переменных, идущих через запятую после форматной строки. Форматная строка позволяет нам *отформатировать* выводимый текст, вставив в него значения нужных переменных.

Слайд 10

Давайте напишем программу, спрашивающую у пользователя радиус бассейна (r) и его глубину (h)

```
#include <stdio.h>
#include <locale.h>
int main(){
setlocale(LC_ALL, "Russian");
const double pi = 3.14;
double V, r, h;
printf("Введите значения r и h через пробел: ");
scanf_s("%lg %lg", &r, &h);
V = pi * r * r * h;
printf("Объем бассейна равен %lg\n", V);
return 0;
}
```

Затем она вычислит объем бассейна по формуле $V = pi \cdot r^2 \cdot h$ и выведет его.

Из данного фрагмента мы видим, что с помощью функции `scanf_s` можно считывать с клавиатуры сразу несколько значений.

Слайд 11

Выражения – это основная рабочая сила языка Си. Выражения бывают двух типов – математические и логические.

Слайд 12

Математическое выражение – *это совокупность имен переменных, символических и литеральных констант, скобок и знаков математических операций (+, -, *, /, %)*порядке приоритетов операций. Приоритеты рассмотренных операций знакомы нам еще со школы*, представляющая собой корректную формулу*. Главная особенность выражений – это то, что они *вычислимы*, то есть их значение можно подсчитать. Вычисляются выражения интуитивно понятным образом: переменные и символические константы заменяются на их текущие значения и получившаяся математическая формула вычисляется в приоритетном порядке. Самый высокий приоритет у скобок, затем идут умножение, деление и взятие остатка от деления, а потом сложение и вычитание.

Помимо перечисленных основных операций +, -, *, / и % язык Си также имеет операцию *унарного минуса*, который инвертирует знак своего операнда

(переменной a будет присвоено значение переменной b с противоположным знаком):
a = -b;
унарного плюса (который, собственно, ничего не делает и существует только для полноты картины), а также специальные операции *префиксного и постфиксного инкремента* «++», увеличивающего значение аргумента на единицу, и *декремента* «--», уменьшающего значение аргумента на единицу.

Слайд 13

Операция	**Поведение**	**Пример**

Постфиксный инкремент	1. Сохраняет значение b в промежуточной временной переменной
	2. Увеличивает на единицу значение переменной b
	3. Возвращает значение промежуточной временной переменной
		`int a, b = 1;`
		`a = b++;`
		`printf("%d %d", a, b);`
		Вывод на экран:
		`1 2`

Префиксный инкремент	1. Увеличивает на единицу значение переменной b
	2. Возвращает значение переменной b
		`int a, b = 1;`
		`a = ++b;`
		`printf("%d %d", a, b);`
		Вывод на экран:
		`2 2`

Декремент ведет себя так же, только не увеличивает, а уменьшает значения соответствующих переменных на единицу.
Так как описанные в таблице операции изменяют значения переменных, над которыми они применяются, то мы не можем
использовать их с константами. В частности, следующая строка вызовет ошибку компиляции:
a = 3++;
потому что для работы инкременту и декременту нужно *леводопу-стимое* выражение (l-value), с которым связана ячейка памяти, а не просто литеральная константа.

Слайд 14

С этими операциями необходимо обращаться осторожно, особенно используя их для одной и той же переменной в составе одного выражения, потому что результат не всегда может оказаться таким, как мы ожидали. Рассмотрим пример:

```
int a, b = 1, c;
a = b++ * b++;
printf("%d %d\n", a, b);
b = 1;
a = ++b * ++b;
printf("%d %d", a, b);
```

Вывод на экран:

```
1 3
9 3
```

Внимательно проанализировав код, приведенный выше, вы поймете, что в нем не все так очевидно. Самое удобное использование инкрементов и декрементов – это когда нам необходимо изменить на единицу какую-то одну переменную. В этом случае, вместо сравнительно длинной конструкции:

```
i = i + 1;
```

можно просто написать

```
++i;
```

При этом лучше (если это не принципиально) использовать именно префиксные версии операций, так как они не создают временных переменных для хранения старых значений операндов.

Слайд 15

Для удобства программиста язык Си предоставляет ряд комбинированных операторов, являющихся связками арифметических операций и оператора присваивания:

```
a += 4;          то же, что и          a = a + 4;
a -= 1;          то же, что и          a = a - 1;
a *= 10;         то же, что и          a = a * 10;
a /= 10;         то же, что и          a = a / 10;
a %= 2;          то же, что и          a = a % 2;
```

Слайд 16

Логическое выражение аналогично арифметическому – оно тоже представляет собой некоторую формулу, состоящую из идентификаторов, констант и символов операций; мы тоже можем его вычислить и получить конкретный результат. Основное отличие состоит в том, что в логических выражениях результат может быть лишь одним из двух: `0` – ложь, `1` – истина. Таким образом, логическое выражение

представляет собой вопрос, на который можно ответить либо «да», либо «нет».

Слайд 17

Еще одним отличием является возможность использования операций *сравнения* и *логических связок*, результатом применения которых также являются истина или ложь. Операций сравнения всего шесть: `<`, `>`, `<=`, `>=`, `!=`, `==`. Следует сразу запомнить, что в отличие от ряда других языков программирования, в языке Си **операция сравнения «==» отличается от операции присваивания «=»**.Плохая новость состоит в том, что это не является синтаксической ошибкой и ваше приложение скомпилируется и запустится, а ошибочный результат может проявить себя совсем нетривиальным образом.

Слайд 18

Логические операции позволяют комбинировать отдельные логические выражения, получая таким образом более сложные «вопросы». Их всего три: `||` – «ИЛИ», `&&` – «И», `!` – «НЕ». Операции «И» и «ИЛИ» – бинарные, «НЕ» – унарная. Для описания работы этих операций обычно пользуются *таблицами истинности*, в которых перечислены все возможные комбинации значений аргументов и соответствующие им значения операций:

x	y	x && y	x	y	x || y	x	!x
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Мы видим, что операция «И» истинна тогда и только тогда, когда оба ее аргумента истинны, операция «ИЛИ» истинна, когда хотя бы один из ее аргументов истинен, а операция «НЕ» просто инвертирует истинность своего аргумента.

Слайд 19

Вычисление выражений идет согласно *приоритетам* операций, представленным в таблице 3.2, при этом самый высокий приоритет, очевидно, у скобок.

Рассмотрим несколько примеров логических выражений.
Пример 1. Пусть есть переменная A. Необходимо записать выражение, которое истинно, если значение A четное.
Решение: (A % 2 == 1).
Пример 2. Для двух переменных A и B записать выражение, которое истинно, когда эти переменные положительны.
Решение: (A > 0 && B > 0).
Пример 3. Даны две переменные A и B. Записать выражение, которое истинно, если эти переменные имеют разные значения.
Решение: (A != B).
Пример 4. Даны три переменные A, B и C. Записать выражение, которое истинно, когда C равно максимальному значению из A и B.
Решение: ((A <= B && C == B) || (A >= B && C == A)).
Во всех вышеперечисленных примерах знаки операций отделены слева и справа пробелами. С точки зрения синтаксиса это не обязательно: мы можем написать A<=B вместо A <= B, но в целях повышения удобочитаемости лучше этого не делать.